applied sciences

MDPI

Article

Evaluation of Classical Machine Learning Techniques towards Urban Sound Recognition on Embedded Systems

Bruno da Silva [1,2,*], Axel W. Happi [2], An Braeken [1] and Abdellah Touhafi [1,2]

[1] Department of Engineering Technology (INDI), Vrije Universiteit Brussel (VUB), 1050 Brussels, Belgium
[2] Department of Electronics and Informatics (ETRO), Vrije Universiteit Brussel (VUB), 1050 Brussels, Belgium
* Correspondence: bruno.da.silva@vub.be; Tel.: +32 2 6293768

Received: 16 July 2019; Accepted: 11 September 2019; Published: 16 September 2019

Abstract: Automatic urban sound classification is a desirable capability for urban monitoring systems, allowing real-time monitoring of urban environments and recognition of events. Current embedded systems provide enough computational power to perform real-time urban audio recognition. Using such devices for the edge computation when acting as nodes of Wireless Sensor Networks (WSN) drastically alleviates the required bandwidth consumption. In this paper, we evaluate classical Machine Learning (ML) techniques for urban sound classification on embedded devices with respect to accuracy and execution time. This evaluation provides a real estimation of what can be expected when performing urban sound classification on such constrained devices. In addition, a cascade approach is also proposed to combine ML techniques by exploiting embedded characteristics such as pipeline or multi-thread execution present in current embedded devices. The accuracy of this approach is similar to the traditional solutions, but provides in addition more flexibility to prioritize accuracy or timing.

Keywords: urban sound classification; machine learning; embedded system; environment sound recognition; audio feature extraction; edge computing.

1. Introduction

Despite the potential application of urban sound recognition in Wireless Acoustic Sensor Networks (WASN), there exists a lack of evaluation of existing solutions. The achievable classifier's accuracy or the demanded time to perform the sound classification are just a couple of parameters, which must be taken into account when considering WASN applications demanding sound recognition. Current embedded systems provide enough computational power to perform urban noise classification, enabling edge-computing solutions. Nevertheless, we believe that such embedded devices are currently underused for such type of sound classification.

The presented work evaluates the most popular Machine Learning (ML) techniques for Environment Sound Recognition (ESR), in particular for urban sound classification. Existing open source libraries for audio analysis and datasets for ESR are used to evaluate the achievable accuracy of classical sound classifiers. The best performing classifiers are evaluated on an embedded system in order to determine the achievable accuracy and the execution time that could be expected on such constrained devices. Several datasets for ESR, which partially include urban sounds are combined here to create larger datasets targeting urban sound classification. In addition, a scalable approach is proposed to not only exploit some characteristics of embedded devices but also to enable the combination of different sound classifiers to prioritize accuracy or response time. The steps followed to construct a hierarchical dataset, the selection of the audio features extracted to fetch the sound classifiers and the parameter's exploration for the optimization of the classifiers are presented in detail.

Contributions towards urban sound recognition on embedded devices:

- A cascade approach for scalable dataset customization.
- Evaluation of classical ML techniques for ESR on an embedded system.

This paper is organized as follows. Section 2 presents related work. The methodology used for the evaluation of the ML techniques is detailed in Section 3. In Section 4, the proposed cascade approach is presented. The experimental results are detailed in Section 5. Finally, our conclusions are presented in Section 6.

2. Related Work

Several authors have compared classical ML techniques for sound classification for the evaluation of their ESR datasets [1–3]. However, a discussion about the convenience of embedding ML techniques for sound classification is missing.

The work presented in [4] is one of the few describing an environmental sound classifier fully embedded on a wearable device. Their embedded system uses a 16-bit microcontroller (MCU) to perform a 1D Haar-like filtering for the feature extraction combined with a Hidden Markov Model (HMM) for the sound classification. Instead, this paper presents a more general evaluation of the most popular sound classifiers in terms of accuracy and response time.

A Convolutional Neural Network (CNN) is proposed as a sound classifier in [5] for Wireless Sensor Networks (WSN) composed of Raspberry Pi acting as a node. The operations on the Raspberry Pi involve the encoding of the audio and its transmission to a computer, where the sound classification is performed. We believe that the feature extraction and the sound classification can be executed in such a relatively powerful embedded device. Our tests include the execution on a Raspberry Pi of all the operations required for the sound recognition.

A distributed sensor network capable of urban sound recognition is presented in [6]. The authors propose a CNN-based classifier able to recognize 14 different types of urban sounds, which is embedded on a Raspberry Pi 3 acting as node of a distributed sensor network. Although all the operations seem to be embedded, no profiling of the execution time nor discussion about the achieved accuracy are provided.

A centralized WSN for ESR is proposed in [7]. Their approach uses distributed nodes with minimal compute power for the audio data acquisition, compression and transmission to a central processing node (Raspberry Pi 3), where a Gaussian Mixture Model (GMM) performs the sound classification. An accuracy of 72% is achieved for only four categories of urban sounds. Unfortunately, the authors do not profile the embedded executions for the sound recognition or provide details of the overall latency of their centralized approach.

The authors in [8] discuss the achievable accuracy and the required modifications to embed on a WSN node their Audio-Visual Correspondence (AVC) based architecture for urban sound recognition. Despite discussing the required modifications on their architecture, only the memory requirements are considered as the constraint parameter of the WSN node. The computational demand of the operations required for the urban sound recognition is not discussed for such contrained devices. Our evaluation covers not only the achievable accuracy, but also the time of response for each classifier.

A system for monitoring, analysis and mitigation of urban noise pollution is proposed in [9,10] as part of the SOunds of New York City (SONYC) project. A Raspberry Pi-based system acting as WSN nodes performs continuous Sound Pressure Level (SPL) measurements while acquiring, compressing and transmitting 10-second audio snippets at random intervals. Although the final objective is to fully embed the urban sound classification in the WSN node, the acquired audio is currently recognized on a laptop machine.

To summarize, some authors have already evaluated different sound classifiers [1–3], but very few have effectively executed them on an embedded system [4,6,7]. Our evaluation not only covers

a larger amount of sound classifiers but also profiles their execution on a state-of-the-art embedded system.

3. Methodology

3.1. Datasets

The automatic urban sound recognition uses ML techniques for the sound classification. Such classifiers need a training period before being able to recognize a particular type of sound. Audio datasets containing labelled urban sounds are required for such training.

3.1.1. Open Source Environmental Sound Datasets

Table 1 summarizes the most popular open source datasets for urban sound and ESR, which are used in our methodology to identify the most suitable sound recognition systems to be embedded. Although most of the categories in these datasets are not directly urban related, they present enough representative urban sounds to be considered for our purpose.

- **BDLib** The authors in [1] created an audio dataset composed of 12 categories representing real-life situations. The collected audio is from sources such as *BBC Complete Sound Effects Library*, *Sony Pictures Sound Effects Series* and the online sound repository called *Freesound* [11]. Each category has 10 audio files of ten seconds each, with great variation between them.
- **UrbanSound** UrbanSound is a large audio dataset of 12 GB with around 18.5 hours of labelled audio [3]. This dataset is exclusively composed of urban sound grouped in 10 categories. Similarly to the BDLib, this dataset is created by manually filtering and labelling every recording from the online sound repository called *Freesound* [11]. Although a larger subset of 4-second audio clips called UrbanSound8K has been also created based on the UrbanSound dataset that is not considered here.
- **ESC** An annotated collection of more than 2000 short audio clips belonging to 50 different categories is presented in [2]. This dataset (ESC-50) comprises not only urban sounds but also present a large number of categories including environmental recordings such as animal sounds, domestic sounds or natural soundscapes. A shorter dataset version recognizing 10 audio categories (ESC-10) is used in this work.

Table 1. Details of the most popular open source datasets for urban sound and ESR.

BDLib Dataset		ESC-10 Dataset		UrbanSound Dataset	
Categories	**Total Time (s)**	**Categories**	**Total time (s)**	**Categories**	**Total time (s)**
Airplane	100	Dog barking	200	Air conditioner	6577
Alarms	100	Baby crying	200	Car horn	4819
Applause	100	Clock tick	200	Children playing	13454
Birds	100	Person sneezing	200	Dog bark	8399
Dogs	100	Helicopter	200	Drilling	4864
Footsteps	100	Chainsaw	200	Engine idling	3604
Motorcycles	100	Rooster	200	Gun shot	7865
Rain	100	Fire cracking	200	Jackhammer	4328
Sea waves	100	Sea waves	200	Siren	4477
Wind	100	Rain	200	Street music	6453
Rivers	100				
Thunderstorm	100				

3.1.2. Limitations

The referred datasets are lacking in the type and number of categories for urban sounds.

- **Large number of categories per dataset**: Whereas the UrbanSound dataset only contains 10 urban sound categories, the ESC-50 dataset contains 50 categories, but many are not audio recordings from the urban environment. A larger number of categories would enable the recognition of more urban sounds.
- **Small number of audio samples per categories**: The datasets do not contain a large number of audio samples, which would benefit the training of the classifiers.
- **Irrelevant categories**: Sound categories such as *sea waves* or *applause* available in the BDLib dataset are not of interest for urban sound recognition.

These limitations can be alleviated by rearranging or combining datasets. Sounds belonging to the same category can be grouped and similar categories can be fused to create new categories. The creation of a new dataset from the existing datasets is detailed in Section 5.4.1.

3.2. Audio Segmentation and Feature Extraction

In our experiments, we were not looking for the best feature/classifier, which led to the best accuracy, but instead were interested in what is the achievable accuracy and the execution time of the classic ML techniques reported in the literature.

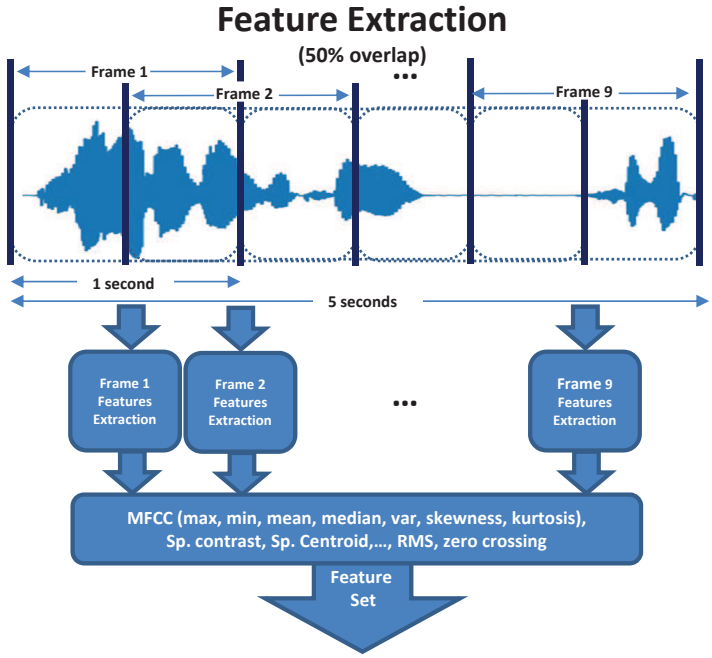


Figure 1. A 5-second input audio signal is split in frames of one second with a 50% overlap. Each frame is sampled at 44.1 kHz and a set of audio features are extracted.

Figure 1 details the audio segmentation and the feature extraction. A HANN window [12] with a length of 44,100 samples and 50% overlap is used to divide every 5-second segment of the audio files into frames. Since the sampling frequency is 44,100 Hz, the duration of each audio frame is one second. The choice of the particular window is based on the fact that the sounds included in the data sets, although non-stationary, do not contain abrupt changes in time. The choice of the sampling frequency is based on [1].

The audio features are extracted per audio frame. The selection of the audio features can be done using software environment tools such as Weka [13]. However, an analysis about the impact in accuracy based on the selection of the features is discussed in Section 5.4.2. The first 12 Mel-Frequency Cepstral Coefficients (MFCC) are extracted per audio frame. The per-frame values of each coefficient are summarized across time using statistics. As a result, each audio frame contains a vector containing:

mean, median, minimum, maximum, variance, skewness and kurtosis. Other features, which are extracted per frame are the spectral contrast, the spectral centroid, the Root-Mean Square (RMS), the spectral bandwidth, the spectral roll off and the zero crossing. This results in a feature vector of 90 elements per frame. All of these feature vectors per frame conform the feature set, which is used by the ML techniques to perform the sound classification.

3.3. Classifiers

There exist several ML techniques, which enable performing sound classifiers. For our evaluation, we have selected supervised ML, which use correct labelled data during their training. The selected supervised ML techniques are:

- k-Nearest Neighbour (k-NN)
- Naive Bayes
- Artificial Neural Network (ANN)
- Support Vector Machine (SVM)
- Decision Tree

These ML techniques are known to deliver good performance when classifying acoustic events. Although several classic ML techniques have been compared in [1] and in [3], there is no evaluation discussing timing results. This is an important aspect because, due to the lack of computational power that embedded devices suffer, some of the evaluated classifiers might be discarded due to their execution time. Our analysis uses a random 80% of the labelled sounds of one dataset for the training and the remaining 20% is used for the validation. While the ratio of correct recognition of that 20% of audio files provide us with the accuracy of the classifier, the time needed by the classifier to individually recognize each sound determines its execution time.

The selected ML techniques have already been evaluated for the sound datasets described in Section 3.1.1 like in [1,2] and in [3]. Although it leaves other ML techniques such as HMMs or Logistic Regression out of our evaluation, we considered that the selected classic ML techniques provide enough diversity for our analysis. CNNs are not considered for a different reason. CNNs have become very popular in the last few years thanks to their use in Deep Learning architectures. Relevant publications evaluating CNNs for sound recognition are [14–17]. For instance, the authors in [17] demonstrate the potential of CNN-based sound classifiers using a limited number of labelled sounds in existing datasets. The discussion about the computational needs of such architecture is expected to be satisfied by general-purpose GPUs. Some authors have already considered constrained embedded devices [8] for CNN-based sound classifiers. The authors in [8] discuss the feasibility of embedding a CNN on a WSN device, but they only evaluate the required memory consumption. Many other computational operations, such as the feature extraction or even the execution of the CNN on such a compute-limited device, are not evaluated. Moreover, despite the use of techniques for data augmentation, the lack of annotated audio data for supervised learning of CNN classifiers remains a bottleneck [9]. We believe that such CNN-based sound recognizers have the potential to overcome other traditional ML techniques for certain sound recognition, but their execution time in current embedded systems is not (yet) faster than classical ML techniques. Although our approach is evaluated in a relatively powerful embedded system such as a Raspberry Pi, capable enough to perform CNN-based classifiers, our ultimate goal is to evaluate the achievable accuracy and time to response that such ML techniques can achieve in general-purpose CPUs similar to ARM-based microprocessors. The proper evaluation of CNNs for sound classification on embedded devices requires an extensive analysis, which is out of the scope of this work.

3.4. Libraries

Operations such as the feature extraction or the classifiers are already supported by many existing libraries. Table 2 presents a list of related audio analysis libraries. Our tests on an embedded system

require the use of compatible libraries, which strongly restrict the options detailed in Table 2. For instance, the Essentia library [18] enables audio analysis and feature extraction, and can be combined with a C++ library called Gaia to implement similarity measures and classification. However, despite being cross-platform, its implementation on ARM-based devices is not yet fully supported since only ARM-based devices running Android O.S. are supported.

Table 2. List of some libraries for audio analysis. The selected library is highlighted in bold.

Name	Language	Description
pyAudioAnalysis [19,20]	Python	Audio Feature Extraction and Classification
Yaafe [21]	Python	Audio Feature Extraction
Essentia [18,22]	C++	Audio Feature Extraction and Classification
aubio [23]	C	Audio Feature Extraction
CLAM [24]	C++	Audio Feature Extraction
LibROSA, [25], [26]	**Python**	**Audio Feature Extraction**
Matlab Audio Analysis Library [27]	Matlab	Audio Feature Extraction and Classification
PyCASP [28,29]	Python	Audio-Specialized library for automatic mapping of computations onto GPUs or multicore CPUs.
seewave [30]	R	Audio Feature Extraction
bob [31,32]	C++/Python	Audio Feature Extraction and Classification

Despite the high execution overhead (related to C for example) introduced by Python, we preferred the use of Python packages due to the existence of an independent Python package scikit-learn 0.21.2 [33] specialized in ML techniques. Although several libraries include their own classifiers, we use thePython package scikit-learn since our evaluation covers several types of sound classifiers. In fact, some libraries such as pyAudioAnalysis base their support in ML techniques by using the scikit-learn package. Regarding the feature extraction, the LibROSA Python package [25] is selected due to supporting ARM-based processors.

4. A Cascade Approach

Urban environments present a rich acoustic context, where many types of sounds can coexist. Traffic noise or the sounds generated by other human activities present a large variety, which might not be properly described in a standalone dataset. Datasets such as ESC-50 [17] present up to 50 different categories of urban noises. Trained classifiers recognizing such number of categories have been shown to achieve relatively low accuracy in recognizing sounds [1,34]. The proposed cascade approach intends to solve such limitation. It consists of a multi-stage classifier such as the one depicted in Figure 2. The audio features are extracted and processed in a two-stages classifier. The first stage classifies the sound in one of the *supercategories*, which correspond to one particular dataset. The second stage uses the same features to determine the category of the sound in the dataset selected in the first stage. This cascade approach enables the use of different types of classifiers at each stage and for each *supercategory* or dataset. Therefore, while the first stage is composed of one single classifier, the second stage is composed of multiple and potentially different types of classifiers trained for each *supercategory* or dataset.

The main advantages of this approach are:

- Different classifiers can coexist in the same system. The optimal classifier is selected based on the recognized sound.
- A multi-stage approach enables the parallel execution of the classifiers. This would lead to a faster sound recognition when exploiting multi-thread or pipeline capabilities of the system.
- Moreover, factors such as their execution time or even power consumption, both critical parameters on embedded systems, can be also considered in the selection of the active classifiers.

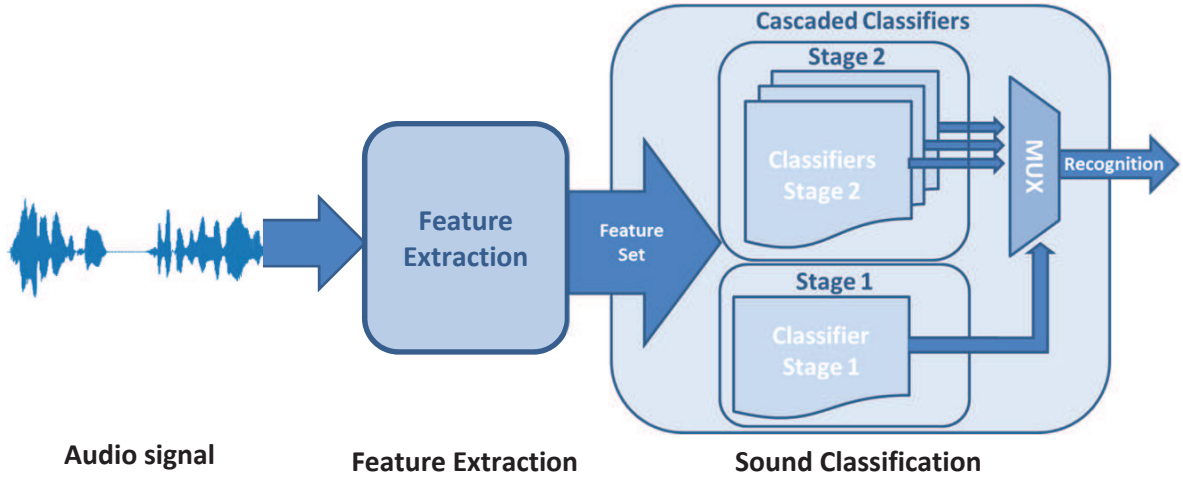


Figure 2. The multi-stage sound classification is composed of two stages, an initial stage performs the broad classification, while a bank of classifiers are used to determine the sound's category.

The cascade approach demands the creation of hierarchical datasets, which can be composed of tens of categories. Figure 3 depicts one example of such hierarchical dataset. The training for the first-stage classifier is done by treating each *supercategory* or dataset as a single dataset category. For instance, categories 1 to 4 of the Dataset 1 in Figure 3 would be considered as one category when training the first-stage classifier. The same approach is applied for the other *supercategories* or datasets. In order to preserve the accuracy, each *supercategory* or dataset must be specialized, containing categories belonging to specific types of sounds. More details about how to build such hierarchical dataset is provided in Section 5.4.1.

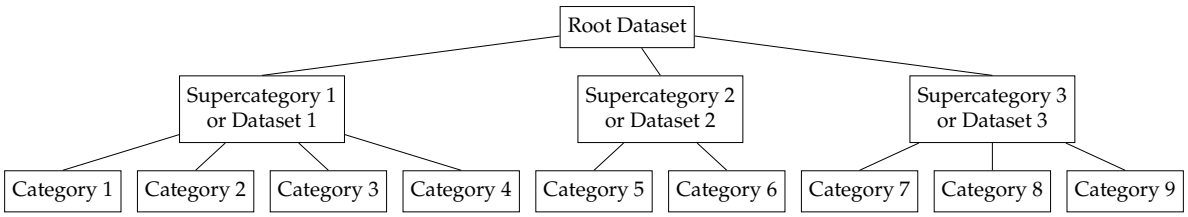


Figure 3. Example of a hierarchical dataset to be used by our cascade approach.

A similar approach has been presented as a part of the challenge proposed for the Workshop on Detection and Classification of Acoustic Scenes and Events (DCASE) [35]. The competitors have to perform a course-grain and/or a fine-grain sound recognition of a two-stage hierarchical dataset composed of seven datasets and a total of 23 categories of urban sounds recorded by the SONYC project [9]. Instead, our proposed cascade approach is a general solution that can be extended by increasing the number of stages and benefits from the combination of different sound classifiers. Although a multi-stages cascade approach increases the operations for the sound classification, it can be exploited by embedded systems. Many ARM-based embedded systems incorporate multi-core processors capable of concurrently executing the classifiers in different threads. The full potential can be achieved in FPGA-based embedded systems since both operations can be executed in the pipeline, resulting in an overlapped execution. The present work evaluates this cascade approach in a multi-core processor. For the sake of simplicity, we only consider a two-stage approach for our experiments.

5. Experimental Results

The experimental results presented in this section are grouped in three main blocks. Firstly, the evaluation of the classifiers for the existing datasets is presented. Secondly, the proposed cascade approach is evaluated. Finally, the best classifiers are evaluated in an embedded system.

The results presented in this section are:

- **Evaluation of the classifiers for the existing datasets**: The LibROSA Python package [25] is used for the audio feature extraction for each dataset discussed in Section 3.4. The evaluation of each selected classifier (k-NN, ANN, Naive Bayes, Decision Tree and SVM) uses 80% of the dataset samples for the training and the remaining 20% for the validation. The performance of the classifiers is compared.
- **Optimization of the classifiers**: The classifiers are optimized per dataset. This optimization is performed by looping the classifiers' parameters on different data configurations and checking for the set of parameters resulting in the smallest mean error. The evaluation of the classifiers on the hierarchical dataset is also done. This evaluation involves different feature sets in order to identify the impact of the feature selection on the recognition's accuracy and the computational speed.
- **Dataset for the cascade approach**: Existing datasets are combined in a hierarchical dataset to be used by the cascade approach.
- **Evaluation of the cascade approach**: The cascade approach is compared to the traditional solution using an equivalent dataset where all the categories of the hierarchical dataset are different categories in the same dataset.
- **Evaluation on an embedded system**: According to the results, the optimal classifier is embedded on a Raspberry Pi. The performance and computational speed of the classifier are then analysed.

5.1. Experimental Setup

Our tests evaluate the existing datasets, the audio features and the classifiers. All the experiments have been firstly executed on an Apple MacBook Pro (2.3 GHz Intel Core i5), in order to evaluate the impact of the features' selection and to compare the achievable performance of the classifiers. The experimental measurements on the embedded system have been obtained after 10 executions on a Raspberry Pi 3B+.

Firstly, the feature extraction is performed by using the Python 2.7 package LibROSA 0.6.3 [25]. The features' selection can be done by importing Weka [13] on Python after converting the features' files in an .arff format. However, in most of our experiments, all the features mentioned in Section 3.2 are used. Secondly, the classifiers are implemented using the Python library scikit-learn 0.21.2 [33]. The classification is performed by splitting the data into 20% testing and 80% training. Each execution of the Python scripts is iterated 10 times to obtain the average of the time extraction per feature. Notice that the feature extraction and the sound classification are operations independent of the approach since both use the same extracted features for the sound classification at all stages. Therefore, our experiments target the profiling of the sound classifiers in terms of accuracy and timing.

5.2. Experimental Timing Profile

The total execution time (t_{exec}) is defined as the sum of the times demanded by the individual operations. The time required for the feature extraction is defined as $t_{extraction}$ and represents the time needed to extract the feature set (Set 3) defined in Table 7. The average time required by the classifier to perform the sound classification using the feature vectors from all frames is defined as $t_{classification}$. This value corresponds to the sum of the time required by each stage i, in case of the cascade approach ($t^{stage_i}_{classification}$), to process the feature vectors from all frames, or, alternatively, it represents the time for the sound classification in case of a single classifier. Therefore, t_{exec} can be defined as follows:

$$t_{exec} = t_{extraction} + t_{classification}, \quad (1)$$

which becomes

$$t_{exec} = t_{extraction} + t^{stage_1}_{classification} + t^{stage_2}_{classification}, \quad (2)$$

in case of a two-level cascade approach.

Although the input audio is read from WAV files for our experiments, the time to read these files is not considered since the sound classification is assumed to be performed from the audio directly coming from a microphone.

5.3. Evaluation of Classifiers per Dataset

5.3.1. Default Classifiers

The reported accuracy of several classifiers is summarized in Table 3. The paper associated with each dataset is selected for this comparison and is to be used as a reference of the expected performance. Nonetheless, not all classifiers are evaluated for each dataset.

Our measurements when evaluating multiple classifiers using the existing datasets are summarized in Table 4. Although only the top 10 features with the highest entropy for each dataset reported by the Weka audio analysis tool could be selected, all the features available in LibROSA have been used as input vectors for the classifiers. Whereas our measurements show that most of the classifiers improve the literature's results (Table 3), our SVM classifier presents a significantly lower accuracy than expected. This lack of accuracy can be associated with the configuration of the SVM classifier since we used the default configuration in our tests and its configuration is not usually reported.

The relation between the number of categories per dataset and the classifier accuracy is exposed in Table 4. The dataset ESC-50 composed of 50 categories [2] results in the lowest classifiers' performance compared to the others. This low performance can also be seen in the number of features whose entropy is null, and, therefore, discarded. As a result, a dataset is recommended to have a limited number of categories. In fact, the number of categories should be as small as possible according to [1,36].

Table 4 also depicts the average execution time of each classifier to process the feature vectors from all frames. The average time computation and accuracy of each classifier is done using 10 iterations. There exists a significant difference between the computational time demanded by each classifier. Whereas ANN only demands few microseconds, k-NN presents the highest demand achieving around two milliseconds. Such disparity must be considered when selecting the classifiers to be evaluated on an embedded system taking into account the constrained nature of these devices.

Table 3. Classifiers' accuracy per dataset reported in the literature.

Classifier	BDLib [1]	ESC-10 [2]	ESC-50 [2]	UrbanSound [3]
k-NN	45.5%	66.7%	32.2%	-
Naive Bayes	45.9%	-	-	-
ANN	54.0%	-	-	-
SVM	53.7%	67.5%	39.9%	≈70%
Decision Trees	-	-	-	≈48%

Table 4. Measured classifiers' accuracy per dataset and average execution time of the classifiers with default configuration. Accuracy and timing values are expressed in percentage and milliseconds, respectively. The values in brackets are the Standard Deviation (SD).

Classifier	BDLib		ESC-10		ESC-50		UrbanSound	
	Accuracy [%]	Timing [ms]	Accuracy [%]	Timing [ms]	Accuracy [%]	Timing [ms]	Accuracy [%]	Timing [ms]
k-NN	59.58 (15.22)	1.98 (0.69)	73.50 (6.26)	1.54 (0.44)	46.90 (4.41)	15.13 (0.59)	42.83 (8.09)	5.91 (0.17)
Naive Bayes	62.08 (14.76)	1.04 (0.06)	69.50 (8.64)	0.95 (0.09)	45.00 (5.01)	5.38 (0.69)	53.50 (5.74)	1.21 (0.15)
ANN	52.92 (12.27)	0.18 (0.02)	67.50 (8.89)	0.19 (0.08)	21.20 (2.82)	0.35 (0.06)	39.17 (4.73)	0.22 (0.01)
SVM	30.00 (8.52)	0.71 (0.19)	40.00 (12.47)	0.73 (0.25)	11.80 (4.05)	0.92 (0.11)	29.17 (4.86)	0.65 (0.07)
Decision Trees	29.17 (10.21)	0.55 (0.07)	39.00 (8.09)	0.54 (0.08)	7.50 (2.37)	0.67 (0.04)	26.50 (4.74)	0.64 (0.05)

5.3.2. Optimized Classifiers

Each classifier supports multiple configurations, which are set by the value of specific parameters. The selection of these parameters which potentially optimizes the classifier's performance is explored here for the k-NN and the ANN classifiers. The parameters under evaluation are:

- The number of neighbors is the core deciding factor for the k-NN classifier. The parameter **n_neighbors** determines the number of nearest neighbors.
- The parameter **hidden_layer_sizes** used by the ANN classifier corresponds to the number of neurons in a layer.

Figure 4 illustrates the optimization process for each configuration. The search for the optimal configuration is performed by empirically exploring the set of parameters giving the smallest mean error. For instance, Figure 5a depicts how the smallest error is obtained when K is equal to 1 for the k-NN classifier. This configuration leads to the highest performance for the classifier k-NN using the dataset BDLib. Similar methodology is applied for the ANN classifier. Figure 5b also depicts the exploration of the size of the hidden layer parameter of the ANN classifier for the dataset BDLib. This exploration enables the selection of the optimal configuration for the classifiers based on the dataset. As a result, the empirical values of the classifiers' parameters which lead to higher accuracy are summarized in Table 5. Notice the dependency between the parameters and the dataset. Not only the audio recordings but also the number of categories determine the optimal configuration of the classifiers.

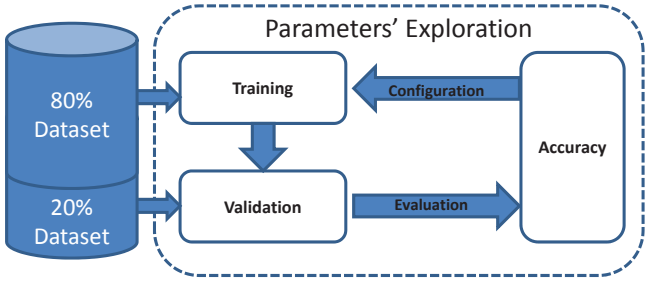
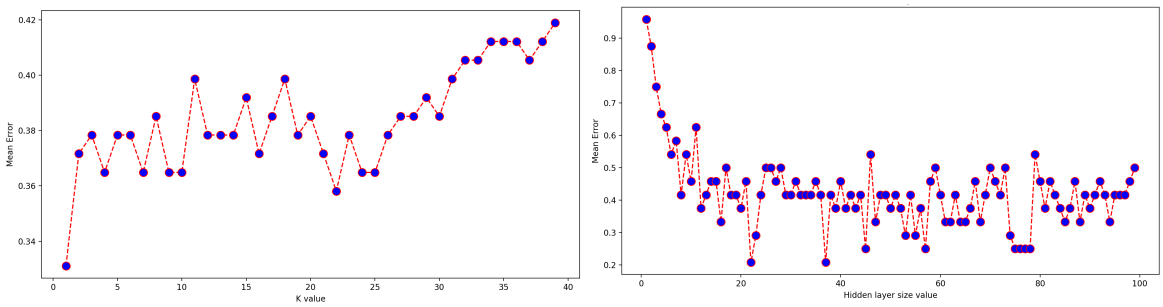


Figure 4. The configuration of the sound classifier is selected based on the achieved accuracy.

Table 5. Optimal values of the classifiers' parameters which minimize the accuracy error per dataset.

Classifier	Parameter	Default	BDLib	ESC-10	ESC-50	UrbanSound
k-NN	n_neighbours	5	1	1	1	1
ANN	hidden layer	10	39	23	87	88



(**a**) K-Nearest Neighbours. (**b**) Artificial Neural Networks.

Figure 5. Exploration of the classifiers' parameters for the BDLib dataset. The k-NN and ANN classifiers are optimized by selecting the value of the parameter leading to the minimal mean error.

Figure 6 shows the achieved accuracy of the optimized classifiers. The features used as input vector for the classifiers are the same as the non-optimized case. As expected, the optimized classifiers significantly increase their accuracy, reaching higher accuracy than detailed in Table 3 for similar comparisons. The evolution of the execution time of the classifiers when optimized is depicted in Figure 7. The ANN classifier demonstrates being more sensitive to optimizations, since it not only increases its accuracy but also significantly increments its execution time for the ESC-50 and the UrbanSound datasets. Nonetheless, this classifier remains significantly faster than the k-NN classifier, as shown in the sections below.

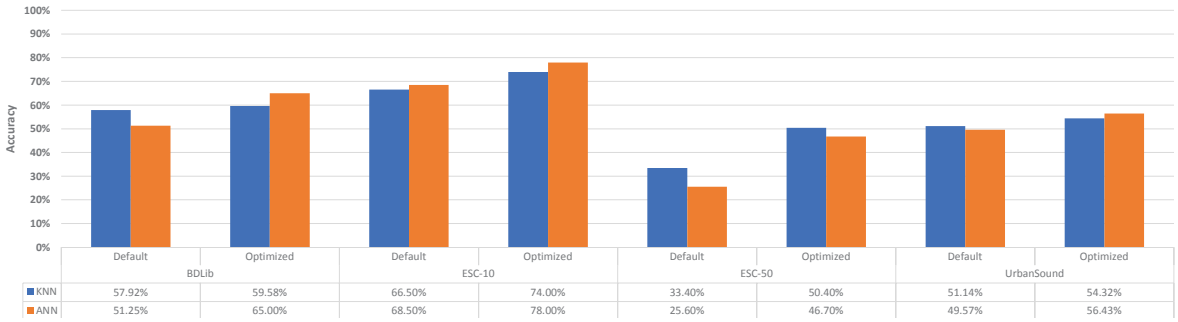


Figure 6. Achieved accuracy of the classifiers with their default and optimized configuration.

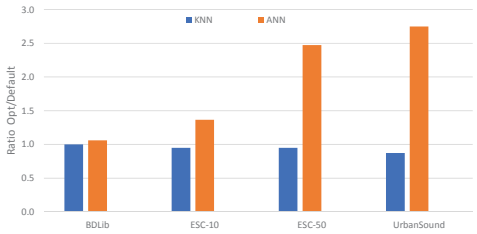


Figure 7. Relative improvement of optimised parameters over default parameters of the classifiers.

5.4. Cascade Approach

In this section, the cascade approach proposed in Section 4 is evaluated. Firstly, a hierarchical dataset is elaborated from existing datasets. Secondly, the best performing audio features are selected. Finally, the classifiers are evaluated for the cascade approach and for an equivalent non-hierarchical dataset.

5.4.1. Hierarchical Dataset

The existing datasets do not necessarily contain the desired categories for a particular application. The generation of new datasets takes effort and is a time-demanding task. Instead, existing datasets can be combined or rearranged in order to satisfy the application's demands.

In order to solve the limitations present on the existing datasets, a semantic hierarchical dataset has been generated (Table 6). The datasets described in Section 3.1.1 are used here to create the new dataset, which is used to evaluate our approach in Section 5.4.4 and in Section 5.5.3. The new dataset is built under certain conditions in order to have higher performance:

- **The dataset must contain relevant categories**: This work intends to evaluate ML techniques in recognizing urban sounds around the campus of the Vrije University Brussels (VUB). Therefore, categories such as *Rivers* or *Sea waves* are discarded. Other similar categories are merged into the same category. For example, the existing categories *Cars* and *Motorcycles* in datasets are merged into a new category *Vehicles*.
- **Small number of categories per data set**: According to [1,34,36], there is an inverse relation between the number of categories contained in a dataset and the classifier's performance. The

more categories, the lower the classifier performance that is obtained. Although the cascade approach intends to minimize this degradation, the number of supercategories should be up to 5 each [36]. Notice that, in spite of up to 25 categories able to be recognized thanks to using the cascade approach, the hierarchical dataset only presents 12 categories of interest.

- **Balanced categories**: All supercategories contained in the hierarchical dataset should be equally balanced. This condition is satisfied by forcing each supercategory to present an equal number of audio samples per category.
- **Sufficient number of audio samples per category**: The data set should have a sufficient number of audio samples per category in order to train the classifier. For the generation of the hierarchical dataset, some online tools such as Freesound [37] and Freesoundeffects [38] have been used to increase the number of audio samples per category.

Table 6. Summary of the categories used in the hierarchical dataset.

Datasets	Categories	Audio data (s)
Root Dataset	Airport	475
	Traffic	475
	Construction	475
	Residence	475
Airport	Airplane	395
	Helicopter	395
Construction	Drilling	170
	Jackhammer	170
Vehicles	Cars	455
	Motorcycles	455
Warning	Cars horn	345
	Siren	345
Residence	Cats	505
	Human	505
	Children	505
	Dogs	505

Figures 8 and 9 illustrate the semantic hierarchical dataset for the cascade approach. This approach uses multiple datasets as categories for the initial stage. The hierarchical dataset is composed of five *supercategories*, which are in fact standalone datasets: *Vehicles*, *Warning*, *Residence*, *Construction* and *Airport*. The hierarchical dataset can be considered as composed of six datasets; these five *supercategories* and the complete dataset without categories (*Root dataset*) used at the initial stage (Figure 8). All are created by combining the datasets described in Section 3.1.1.

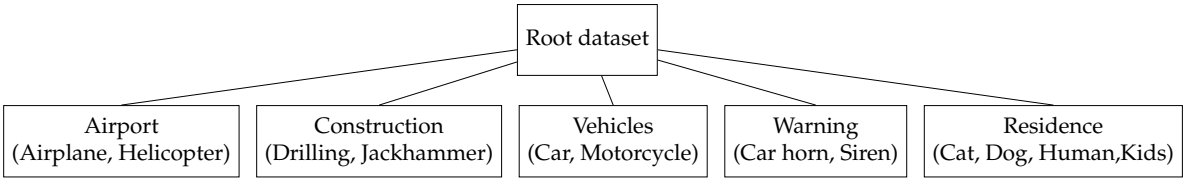


Figure 8. Training dataset for the first stage of the cascade approach. Notice that different sound categories (in brackets) are grouped in the same category.

Based on the prediction of the first stage, a second stage recognizes, the categories of the recognized *supercategory* to which the sound belongs (Figure 9). As a result, the cascade approach supports an overall large number of categories while presenting a small number of categories per dataset.

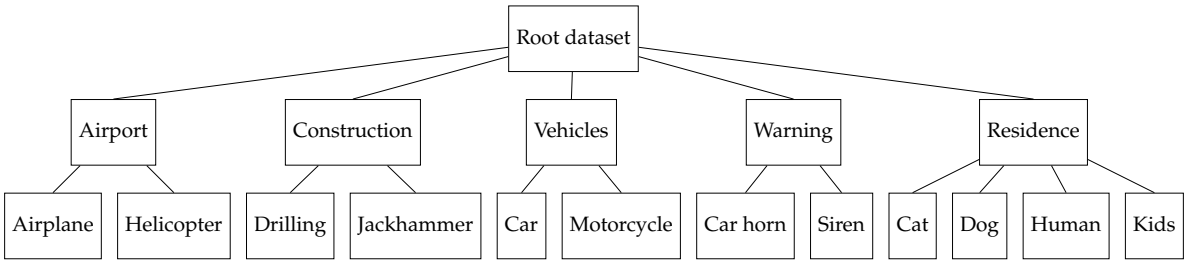


Figure 9. Detailed hierarchical dataset for the cascade approach.

5.4.2. Feature Selection

The selection of the audio features to be used by the classifiers can be done using the Weka audio analysis tool. The Weka's features' selection uses the ranker algorithm *InformationGain*, which gives the set of most relevant features per dataset. This information can be used to select the best feature set leading to the highest classifier performance [1]. The main concern is to have the highest performance while using the smallest number of features as possible. Similar to [1], three sets of features have been created and used to identify the feature set leading to a higher classifier's performance. Each set has been created based on:

- **Set 1** : Features that are common to all datasets.
- **Set 2** : Features that are common to at least three datasets.
- **Set 3** : All features provided by LibROSA.

Table 7 summarizes the features composing the sets under evaluation. The common features to all datasets of the hierarchical dataset are expected to be the most useful. Nevertheless, it resulted in a very short set composed of only two features. Our analysis shows that more features are in common between several datasets, and, in order to increase the number of features to be evaluated, Set 2 and Set 3 are proposed. Whereas Set 2 is composed of features that are also common to at least three datasets, Set 3 is expected to deliver the best results since all supported features are used. Although a more fine-tuning selection of the features can be done by evaluating the linear correlation among the features, as done in [1], we have decided to only evaluate the feature sets depicted in Table 7 for the sake of simplicity.

Table 7. Considered feature sets and average time required for the features' extraction. The values in brackets are the Standard Deviation (SD).

Set 1		**Set 2**		**Set 3**	
Features	**Time [ms]**	**Features**	**Time [ms]**	**Features**	**Time [ms]**
Mfcc 1, 6	17.30 (1.08)	Zero Crossing	4.17 (0.37)	Mfcc 0 - Mfcc 12	17.30 (1.08)
Total time ($t_{extraction}$)	17.30 (1.08)	Sp. Contrast 0, 2	24.58 (1.92)	Sp. Contrast 0 - Sp. Contrast 5	24.58 (1.92)
		Mfcc 0, 2, 4, 5, 7	17.30 (1.08)	Sp. Centroid	12.75 (0.54)
		Total time ($t_{extraction}$)	46.06 (0.62)	Sp. Roll off	12.93 (0.55)
				Sp. Bandwidth	15.84 (0.59)
				Rms	63.95 (3.70)
				Zero Crossing	4.17 (0.37)
				Total time ($t_{extraction}$)	151.55 (1.67)

Table 8 summarizes the classifiers' performance (with default configuration) for each feature set. The values in bold correspond to the highest performance reached for one feature set. The classifiers k-NN, Naive Bayes and ANN achieve higher performance for most of the datasets. Most of the classifiers increase their performance when increasing the number of features. Notice that it is not always the case. For instance, the k-NN and the Decision Tree classifiers do not present significant difference on its performance for the *Construction* dataset. Nonetheless, for most of the cases, Set 3 achieves the highest performance. As a result, this feature set is used hereby for our evaluations.

Table 8. Average accuracy in percentage per data set. The best classifier per data set is highlighted in bold.

Datasets	Sets	k-NN	ANN	Decision tree	SVM	Naive Bayes
Root dataset	Set1	35.61	**52.84**	39.93	25.54	49.19
	Set2	**55.47**	51.76	44.46	33.04	47.03
	Set3	**62.16**	62.09	48.92	45.68	54.46
Airport	Set1	**73.44**	72.81	70.31	61.88	73.13
	Set2	74.38	76.25	77.50	65.31	**78.13**
	Set3	**81.56**	81.25	70.94	74.69	69.38
Construction	Set1	70.00	72.78	72.78	66.67	**81.11**
	Set2	71.11	**85.56**	64.44	74.44	87.22
	Set3	70.56	82.78	62.22	68.89	**89.44**
Residence	Set1	44.69	**69.38**	59.38	37.38	62.88
	Set2	65.56	68.25	64.00	51.38	**68.63**
	Set3	**78.52**	78.88	66.38	71.00	76.25
Vehicles	Set1	66.22	64.32	**67.03**	55.14	66.76
	Set2	72.16	**78.65**	69.19	59.46	72.43
	Set3	77.30	**80.27**	68.92	73.24	77.84
Warning	Set1	60.36	**64.64**	60.36	59.64	64.64
	Set2	66.07	**70.00**	58.93	65.00	63.21
	Set3	65.71	**72.86**	65.00	65.36	65.36

The average execution time of each classifier for each features set is depicted in Figure 10. On the one hand, the classifier k-NN demands a computational time significantly higher than other classifiers, while the ANN classifier demonstrates being extremely fast independently of the number of features. On the other hand, not all classifiers present the same sensitivity to the feature's set.

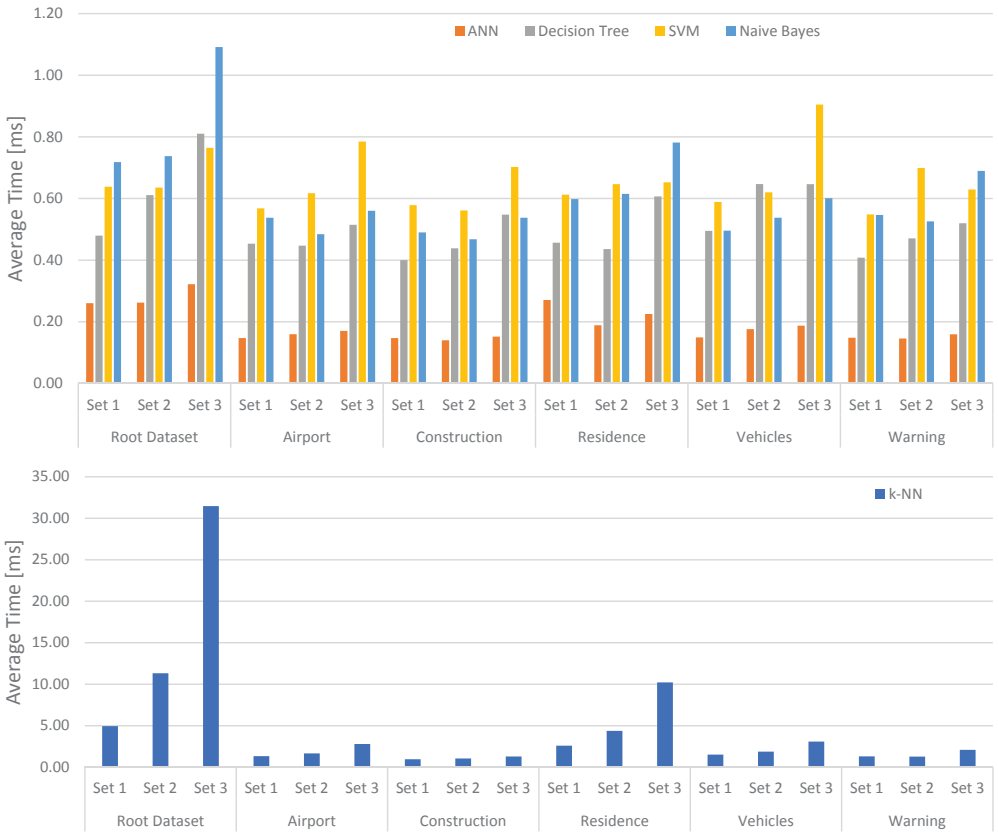


Figure 10. Average timing of the classifiers running with their default configuration.

One would expect that, for most of the classifiers, the increment of the number of features used by the classifiers would lead to more accurate results, but also to slower classifiers. The results summarized in Table 8 and Figure 10 confirm this assumption. Few exceptions exist. For instance, the classifier Naive Bayes reaches a higher performance using Set 1 instead of Set 2 for the *Root* and the *Warning* datasets. Moreover, the execution time of the ANN classifier slightly varies in most of the cases when enlarging the number of features.

5.4.3. Classifier Selection

The evaluated classifiers support multiple parameters, which directly affects their performance. The strategies applied in Section 5.3.2 to optimize the classifiers are applied here to select the best performing classifiers for the cascade approach. As shown in Table 8, the k-NN and the ANN classifiers offer the higher accuracy for most of the datasets of the cascade approach. Although the Naive Bayes classifier could be also considered, this classifier presents a relatively low accuracy for the *Root* dataset, for which accuracy is critical for the cascade approach as discussed in the section below. For these reasons, only the K-NN and the ANN classifiers are selected to be optimized. Table 9 summarizes the default and optimized parameters for the k-NN and ANN classifiers.

Table 9. The parameters for the optimization of the k-NN and ANN classifiers for the datasets of the cascade approach and for the non-cascade approach.

	Optimized Parameters	
Datasets	**n_neighbors**	**hidden_layer_sizes**
Root Dataset	1	96
Airport	1	14
Construction	2	5
Residence	1	48
Vehicles	1	60
Warning	3	48
Non-cascade	1	74

Figure 11 depicts the achieved accuracy when using the default and the optimized configuration of the classifiers using Set 3. As summarized in Table 9, each classifier is optimized to achieve the highest accuracy per data set. The optimization is evaluated using all the available features (Set 3). The classifier ANN presents a higher increment when optimized, offering the highest performance for most of the datasets.

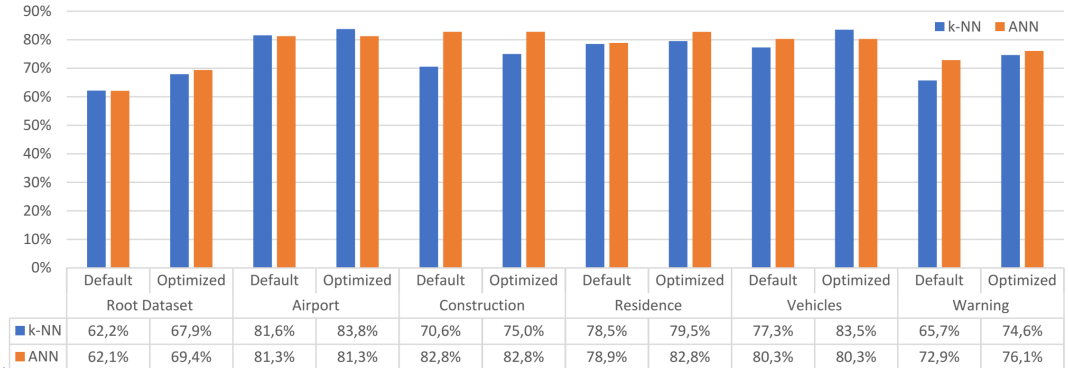


Figure 11. Achieved accuracy of the classifiers with their default and optimized configuration.

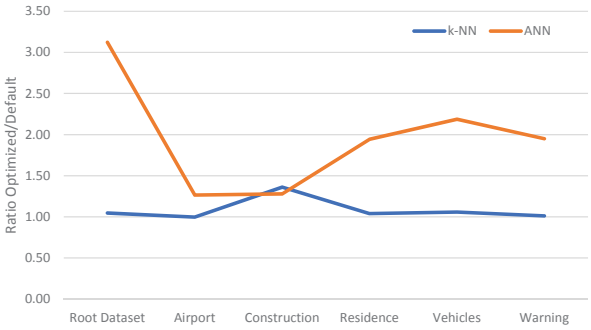


Figure 12. Average execution time of the classifiers based on their configuration.

Figure 12 depicts the ratio between the average execution time when using the default and the optimized configuration of the classifiers. Again, an ANN classifier increments their execution time when optimized for the Root dataset. In general, the execution time does not present any significant variation when optimizing.

5.4.4. Comparison

The proposed approach is compared to a conventional non-cascade approach, which uses a non-hierarchical dataset containing the same categories of each individual dataset of the cascade approach.

The global accuracy of the cascade approach ($Accuracy_{Overall}$) is obtained by multiplying the achieved accuracy at each stage, as defined in Equation (3).

$$Accuracy_{Overall} = Accuracy_{Stage_1} \times Accuracy_{Stage_2}, \quad (3)$$

where $Accuracy_{Stage_i}$ is the accuracy achieved at stage $i \in \{1, 2\}$.

The measured accuracy and timing per classifier for the cascade approach is summarized in Table 10. As expected, the most accurate classifiers are the k-NN and the ANN classifiers. Notice that the Naive Bayes classifier outperforms these classifiers for the *Construction* dataset but offers lower accuracy for most of the datasets. Moreover, due to its relatively low accuracy for the *Root* dataset, this classifier is not a candidate for the initial stage. The overall accuracy of the cascade approach for the k-NN and ANN classifiers obtained by applying Equation (3) is summarized in Table 11. The results follow the achieved accuracy depicted in Figure 11. While the k-NN classifier achieves a slightly lower accuracy than the ANN classifier, it is approximately 30 times slower. Notice how $Accuracy_{Stage\ 1}$ impacts the overall accuracy. The ANN classifier remains the most accurate one. However, classifiers can be combined to increase the overall accuracy or for speed. The last columns in Table 11 depict the accuracy and the timing when combining the most accurate and the fastest classifiers. For instance, to increase accuracy, the ANN can be used for most of the datasets while using k-NN for the *Airport* and *Vehicles* datasets. Such a combination achieves around 58% accuracy but doubles the timing of the ANN classifier standalone. On the other side, a faster sound classification can be obtained when combining the faster classifiers such as using the Decision Tree classifier for the *Root dataset* while using the ANN classifier for the rest of the datasets. The execution time decreases to around 1 ms while the accuracy drastically decreases to around 39%. Both are good examples of one of the benefits of the cascade approach, which enables the combination of classifiers based on the priorities of the application performing the sound recognition.

Table 10. Achievable accuracy and demanded execution time for the proposed cascade approach of the classifiers. The most accurate classifier per dataset is marked in bold. The Standard Deviation (SD) is in brackets.

	k-NN		ANN		Decision Tree		SVM		Naive Bayes	
	Accuracy [%]	**Time [ms]**	**Accuracy [%]**	**Time [ms]**	**Accuracy [%]**	**Time [ms]**	**Accuracy [%]**	**Time [ms]**	**Accuracy [%]**	**Time [ms]**
Root Dataset	67.91 (3.29)	32.94 (1.74)	**69.39 (2.78)**	1.01 (0.17)	48.92 (6.49)	0.72 (0.133)	45.68 (6.43)	0.90 (0.37)	54.46 (2.89)	1.42 (0.85)
Airport	**83.75 (4.11)**	2.78 (0.65)	79.06 (5.11)	0.22 (0.05)	70.94 (7.07)	0.70 (0.15)	74.69 (6.66)	0.80 (0.18)	69.38 (7.91)	0.65 (0.16)
Construction	75.00 (9.53)	1.75 (0.58)	81.11 (6.52)	0.19 (0.10)	62.22 (9.36)	0.53 (0.14)	68.89 (9.14)	0.70 (0.13)	**89.44 (9.61)**	0.59 (0.11)
Residence	79.50 (3.01)	10.61 (0.33)	**82.75 (4.32)**	0.44 (0.06)	66.38 (3.84)	0.91 (0.48)	71.00 (3.98)	0.76 (0.26)	76.25 (7.09)	0.87 (0.09)
Vehicles	**83.51 (4.84)**	3.27 (0.41)	80.00 (8.56)	0.41 (0.25)	68.92 (9.38)	0.57 (0.10)	73.24 (4.11)	0.83 (0.34)	77.84 (8.04)	0.60 (0.07)
Warning	74.64 (6.62)	2.11 (0.24)	**76.07 (4.47)**	0.31 (0.06)	65.00 (8.72)	0.64 (0.15)	65.36 (5.84)	0.99 (0.74)	65.36 (10.79)	0.58 (0.03)

Table 11. Overall accuracy and timing in milliseconds of the cascade approach for the optimized k-NN and ANN classifiers. The average values are marked in bold.

Feature Set	**Path**	**k-NN**		**ANN**		**Most Accurate**		**Fastest**	
		Overall Accuracy [%]	**Time [ms]**	**Overall Accuracy [%]**	**Time [ms]**	**Accuracy [%]**	**Time [ms]**	**Accuracy [%]**	**Time [ms]**
Set 3	**Root Dataset => Airport**	56.87	35.73	54.86	1.22	58.12	3.79	38.68	0.93
	Root Dataset => Construction	50.93	34.69	56.28	1.20	62.07	1.60	39.68	0.91
	Root Dataset => Residence	53.98	43.56	57.42	1.44	57.42	1.44	40.48	1.15
	Root Dataset => Vehicles	56.71	36.21	55.51	1.41	57.95	4.28	39.14	1.12
	Root Dataset => Warning	50.69	35.05	52.79	1.32	52.79	1.32	37.21	1.03
	Mean	**53.84**	**37.05**	**55.37**	**1.32**	**57.67**	**2.48**	**39.04**	**1.03**
	SD	2.99	3.69	1.73	0.11	3.30	1.43	1.22	0.11

Table 12. Classifiers' accuracy and timing using an equivalent non-cascade dataset. The Standard Deviation (SD) is in brackets.

Categories	Feature Set	k-NN		ANN		Decision Tree		SVM		Naive Bayes	
		Accuracy [%]	Timing [ms]	Accuracy [%]	Timing [ms]	Accuracy [%]	Timing [ms]	Accuracy [%]	Timing [ms]	Accuracy [%]	Timing [ms]
Airplane Helicopter Jackhammer Drilling Cats Dogs Humans Children Cars Motorcycles Car horn Siren	Set 3	52.264 (3.88)	16.301 (0.82)	56.981 (6.03)	0.804 (0.17)	28.869 (3.91)	0.857 (0.43)	32.169 (6.21)	1.063 (0.65)	48.962 (4.48)	1.752 (0.14)

Table 12 shows the results of the classifiers' accuracy when all the categories are regrouped in one non-hierarchical dataset. The achieved accuracy of the cascade approach is similar to the traditional approach in all categories. The similarity in the accuracy of both approaches is due to the fact that the non-hierarchical dataset only considers 12 categories. The main advantage of the cascade approach is the additional flexibility due to combining different types of classifiers.

The disadvantage of the cascade approach is the execution time when only using the same type of classifier for all stages. Although the feature extraction is common for both approaches, with values rounding 150 ms for Set 3 (Table 7), the cascade approach requires the computation of two classifiers. Based on the classifier, the timing reported in Table 11 ranges from around 1.2 ms to more than 40 ms for the ANN (Root Dataset => Airport) and for the k-NN (Root Dataset => Residence) classifier, respectively. Due to the limited computational resources, the overall execution on the embedded system is expected to be significantly higher. There is a trade-off between time and accuracy. Additional stages increase the supported number of categories but also the overall execution time. Nonetheless, the ultimate objective is to embed this cascade approach in an FPGA, where it can be computed in pipeline, or in an embedded device with a multi-core processor, where each stage is executed concurrently in a different thread.

5.5. Evaluation on an Embedded System

The evaluated classifiers and the proposed cascade approach have been implemented on Raspberry Pi 3B+ running a Raspbian 4.14 environment. Firstly, the accuracy and timing measurements done on the Raspberry Pi of the five classifiers are discussed. Secondly, the proposed approach is evaluated and compared to a traditional solution running on this embedded system. The libraries and the source code running on the embedded system is the same as the one used for the experiments on the computer. The execution time of each operation related to the feature extraction and to the sound classification has been individually measured. Our experiments are repeated 10 times in order to obtain the average timing and the standard deviation (SD) of the operations.

5.5.1. Feature Extraction on an Embedded System

The measurement of $t_{extraction}$ has been individually obtained. The value of $t_{extraction}$ can be added to the timing of the sound classifiers to obtain the t_{exec} when acquiring the audio data directly from a microphone (Equation (1)). Table 13 summarizes the timing for the different types of audio features running on the embedded system. The computational effort to extract Set 3 requires a lot of time, becoming around 20 times slower than on the laptop (Table 7).

Table 13. Average time required for the feature extraction on the Raspberry Pi. The values in brackets are the Standard Deviation (SD).

Features	Time [ms]
Mfcc 0 - Mfcc 12	622.22 (163.71)
Sp. Contrast 0 - Sp. Contrast 5	336.39 (64.87)
Sp. Centroid	300.24 (52.81)
Sp. Roll off	315.79 (61.37)
Sp. Bandwidth	349.83 (63.33)
Rms	1089.10 (260.62)
Zero Crossing	64.22 (15.19)
Total time ($t_{extraction}$)	3077.79 (125.21)

5.5.2. Evaluation of Classifiers per Dataset Running on an Embedded System

Table 14 summarizes the measured accuracy and timing of the five classifiers for the BDLib, ESC-10, ESC-50 and the UrbanSound datasets running on the Raspberry Pi. The performance of the classifiers is not affected by the platform, achieving similar accuracy to the one depicted in Table 4. As expected, the execution time needed for the sound classification is significantly affected, becoming around 10 times higher in many cases.

Table 14. Experimental values obtained in the Raspberry Pi. The classifiers' accuracy per dataset and the average execution time of the classifiers with default configuration are measured. Timing values are expressed in milliseconds. The values in brackets are the Standard Deviation (SD).

Classifier	BDLib		ESC-10		ESC-50		UrbanSound	
	Accuracy [%]	**Timing [ms]**	**Accuracy [%]**	**Timing [ms]**	**Accuracy [%]**	**Timing [ms]**	**Accuracy [%]**	**Timing [ms]**
k-NN	59.17 (4.73)	12.83 (8.05)	73.50 (10.01)	9.54 (3.39)	48.70 (4.19)	114.53 (5.94)	50.48 (2.91)	80.70 (33.21)
Naive Bayes	68.33 (10.24)	12.42 (5.29)	69.00 (12.87)	10.11 (3.27)	42.10 (3.48)	82.18 (4.42)	49.29 (2.62)	11.85 (4.87)
ANN	50.83 (9.38)	2.20 (0.84)	64.50 (10.12)	1.85 (0.57)	20.90 (4.39)	5.95 (0.29)	50.6 (3.73)	4.34 (1.72)
SVM	30.00 (8.96)	5.01 (0.22)	36.00 (12.65)	4.94 (0.31)	9.40 (3.89)	16.79 (1.07)	25.29 (7.73)	6.44 (2.52)
Decision Trees	29.58 (8.43)	4.36 (0.21)	38.00 (11.34)	4.42 (0.24)	6.80 (2.48)	4.85 (0.35)	34.7 (2.09)	6.14 (2.41)

5.5.3. Cascade Approach on an Embedded System

Like the experiments on the computer, the embedded implementations use the Python 2.7 libraries LibROSA and scikit-learn for the feature extraction and the sound classification, respectively. The training of all six datasets (Root dataset plus the 5 datasets) required for the cascade approach required around 1.5 hours on the Raspberry Pi. This training time is possible by specifying the audio length when loading the audio with LibROSA, which for our case is 5s per audio file. Both the features extraction and classification are also performed on the Raspberry Pi. The ANN and k-NN classifier has been evaluated due to their relatively high accuracy. They are configured with the parameters present on Table 9.

Table 15 summarizes the measured accuracy and timing of the classifiers when using the cascade approach on the Raspberry Pi. The results are the average of 10 executions. Despite the relatively high accuracy achieved by both classifiers for each dataset (or supercategory), the global accuracy when considering the combined accuracy of both classifiers (Table 16) decreases similarly to the accuracy depicted in Table 11. The achievable accuracy obtained by Equation (3) rounds to 54.4% for the ANN classifier and to 50.3% for the k-NN classifier. Both classifiers slightly decrease their accuracy in their embedded version. Some improvements in the dataset (or supercategory) *Warning* would certainly lead to a higher accuracy since both classifiers significantly decrease their accuracy for this dataset. Although the execution time of both classifiers remains in the order of milliseconds, the ANN classifier has increased its timing significantly, becoming only around 3.5 times faster than the k-NN classifier. Notice that the k-NN and ANN classifiers are significantly slower than the other classifiers for the *Root* dataset. There are several reasons for this:

- The k-NN classifier is the most time demanding classifier, not only when embedded (Tables 4, 11 and 12 and in Figure 10). Similarly to other classifiers, the k-NN classifier becomes around 10 times slower when embedded.
- The exception is the ANN classifier. The increment of the execution time of the ANN classifier when embedded, reflected in Table 15, becomes around 100 times slower when compared to Table 11.
- The k-NN and the ANN classifiers are evaluated with their optimal configuration. This fact increments the ANN's execution time like that shown in Figure 7 and in Figure 12.

The main benefit of the cascade approach is the combination of classifiers. Whereas the ANN classifier outperforms the k-NN classifier for the Root dataset, the accuracy of the k-NN remains higher for the *Airport* and the *Vehicles* datasets. Thus, the ANN classifier can be used for all the datasets except the *Airport* and the *Vehicles* datasets, where the k-NN classifier could be applied due to its higher accuracy. Such combination reaches an accuracy of 55.7% but demands 95.7 ms. Note that the overall execution time of the cascade approach decreases if the Decision Tree classifier is used in stage 1 and datasets *Residence*, *Vehicles* and *Warning* in the second stage while the ANN classifier is used for the *Airport* and *Construction* datasets in the second stage. Such a solution would only require 8.6 ms, but the accuracy decreases to 35.7%. Both are examples of how the cascade approach enables multiple combinations, either targeting accuracy or a fast response.

The evaluation of the ANN and k-NN classifiers for the non-cascade approach is summarized in Table 17. The achieved performance of both classifiers is similar to the cascade approach, while the execution time of both classifiers is slightly lower than the two-stages approach. The timing increases from a factor of 8.8 to 52.2 times for the k-NN and for the ANN classifiers respectively compared to the timing when both classifiers are not embedded (Table 12).

Table 15. Achievable accuracy and demanded execution time for the proposed cascade approach of the classifiers running on the Raspberry Pi. The most accurate classifier per dataset is marked in bold.

	k-NN		ANN		Decision Tree		SVM		Naive Bayes	
	Accuracy [%]	**Time [ms]**	**Accuracy [%]**	**Time [ms]**	**Accuracy [%]**	**Time [ms]**	**Accuracy [%]**	**Time [ms]**	**Accuracy [%]**	**Time [ms]**
Root Dataset	66.23 (3.16)	300.18 (33.76)	**68.63 (3.52)**	83.66 (33.62)	47.26 (5.39)	5.03 (0.32)	51.30 (5.01)	7.26 (0.54)	53.22 (3.86)	15.86 (0.98)
Airport	**85.31 (4.89)**	15.53 (2.42)	83.75 (5.67)	2.66 (0.88)	74.06 (9.20)	5.46 (2.29)	76.25 (5.92)	4.93 (1.60)	73.44 (6.95)	5.84 (0.55)
Construction	69.44 (6.28)	7.21 (0.17)	76.11 (9.39)	1.11 (0.06)	61.67 (10.68)	5.85 (2.03)	68.89 (11.15)	5.80 (2.10)	**83.33 (7.35)**	5.17 (0.17)
Residence	75.25 (3.63)	77.34 (4.27)	**82.75 (3.52)**	15.38 (1.19)	70.50 (6.85)	4.72 (0.31)	68.63 (8.16)	6.20 (1.76)	74.13 (3.82)	9.58 (0.49)
Vehicles	**83.51 (6.17)**	18.63 (2.37)	82.97 (8.06)	9.32 (0.44)	78.11 (8.00)	4.49 (0.29)	62.97 (9.01)	4.32 (0.21)	75.95 (8.49)	5.97 (0.41)
Warning	66.07 (12.05)	11.20 (2.22)	**70.71 (11.64)**	5.71 (0.22)	69.29 (7.37)	4.91 (1.57)	62.50 (8.79)	4.84 (1.49)	63.93 (8.98)	5.64 (0.39)

Table 16. Achievable accuracy and demanded execution time of the optimized classifiers performance using the proposed cascade approach running on the Raspberry Pi. The last row details the mean and the standard deviation (SD) of the global accuracy and timing.

Feature set	**Path**	**k-NN**		**ANN**		**Most Accurate**		**Fastest**	
		Overall Accuracy [%]	**Time [ms]**	**Overall Accuracy [%]**	**Time [ms]**	**Accuracy [%]**	**Time [ms]**	**Accuracy [%]**	**Time [ms]**
Set 3	**Root Dataset => Airport**	56.50	315.71	57.48	86.32	58.55	99.19	39.58	7.69
	Root Dataset => Construction	46.00	307.39	52.24	84.77	57.19	88.83	35.97	6.14
	Root Dataset => Residence	49.84	377.52	56.79	99.04	56.79	99.04	33.32	9.75
	Root Dataset => Vehicles	55.31	318.81	56.94	92.97	57.32	102.29	36.91	9.52
	Root Dataset => Warning	43.76	311.37	48.53	89.37	48.53	89.37	32.74	9.94
	Mean	**50.28**	**326.16**	**54.40**	**90.49**	**55.68**	**95.74**	**35.71**	**8.61**
	SD	5.59	29.03	3.90	5.71	4.05	6.21	4.90	1.61

Table 17. Achievable accuracy and demanded execution time of the optimized classifiers performance using the non-cascade approach running on the Raspberry Pi. The Standard Deviation (SD) is in brackets.

Categories	FeatureSet	k-NN		ANN		Decision Tree		SVM		Naive Bayes	
		Accuracy [%]	Timing [ms]	Accuracy [%]	Timing [ms]	Accuracy [%]	Timing [ms]	Accuracy [%]	Timing [ms]	Accuracy [%]	Timing [ms]
Airplane Helicopter Jackhammer Drilling Cats Dogs Humans Children Cars Motorcycles Car horn Siren	Set 3	**52.26**(4.53)	144.09 (59.09)	**54.91**(4.42)	41.85 (18.15)	27.55 (4.62)	5.41 (2.09)	35.85 (5.43)	7.69 (0.14)	49.53 (4.34)	27.33 (10.53)

5.6. Discussion

The measured accuracy decreases for all classifiers when they are embedded. This fact can be related to the limitation of the embedded version of the LibROSA library, on which the audio feature extraction perform differently. Based on our analysis, the expected accuracy of both approaches ranges around 50%–60% for the recognition of different sound categories. It represents a lower range than the CNN-based solutions proposed in [35], where accuracy ranges around 50%–75% for the fine-grain sound recognition of a hierarchical dataset. Although the limited computational power available on embedded devices challenges the use of such CNN-based solutions, recent new commercial devices such as Googles' Edge Tensor Processing Unit (TPU) [39] facilitate the use of this technique for edge computing.

Our measurements show that the most time-demanding operation is the audio features extraction, an operation which requires several seconds on an embedded system. Although this time can be reduced by lowering the number of extracted features, our experiments demonstrate that it certainly affects the accuracy of the sound classifier. Further analysis is needed when selecting the audio features for a particular dataset since not all sound classifiers are equally affected when reducing the features set. Nonetheless, such analysis is beyond the scope of this paper.

Our analysis provides valuable information about the achievable accuracy and the required execution time on embedded devices. Some discussion is needed when targeting real-time sound classification. The frame size of 1 second must be reduced to a few milliseconds and the sound classification might be performed per frame. Our results provide timing information, which can be used to select frame size, determining the time slot to perform the feature extraction and the sound classification. For instance, a frame size of around 100 ms would only provide enough time for the ANN classifier in case of the cascade approach (Table 16), but not for the feature extraction, which for Set 3 rounds to 3 s (Table 13). A frame size of around 700 ms could be enough by only using the features from the MFCC and the ANN classifier in case of the non-cascade approach (Table 17). Although we do not propose a solution for real-time sound recognition on an embedded device, our evaluation can be used for the selection of the frame size, the feature extraction or the classifiers.

Embedded devices are often power-constrained devices. A power analysis of the different approaches would certainly help to not only select the most power-efficient technique but also to help in the selection of the embedded device. One would expect that the cascade approach presents a higher power consumption due to using two sound classifiers. However, the flexibility of this approach enables the combination of power-efficient classifiers, which can lead to low power solutions. A detailed power analysis of the feature extraction and the sound classification is one of the priorities for future work.

6. Conclusions

Our experimental results demonstrate that classical sound classifiers for urban sound recognition achieve a slightly lower accuracy when embedded on an Raspberry Pi 3 at the cost of an increment of the timing. Although the ANN classifier remains faster than the k-NN classifier, its execution time increases by a factor of 100 when embedded, while other ML classifiers increase their execution time by 10 times. The proposed cascade approach obtains similar accuracy to traditional solutions, while providing additional flexibility to prioritize accuracy or timing. Our analysis provides valuable information about the achievable accuracy and the required execution time on embedded devices, which can be used to decide parameters such as the frame size in order to achieve real time. Embedded devices such as the evaluated Raspberry Pi 3B+ demonstrate being powerful enough to achieve urban sound recognition in a reasonable time.

Author Contributions: Conceptualization, B.d.S.; Formal analysis, B.d.S. and A.W.H.; Investigation, A.W.H.; Methodology, B.d.S. ; Software, A.W.H.; Supervision, A.B and A.T.; Validation, B.d.S.; Writing—original draft, B.d.S.; Review and editing, A.B; Funding Acquisition, A.B and A.T.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Bountourakis, V.; Vrysis, L.; Papanikolaou, G. Machine learning algorithms for environmental sound recognition: Towards soundscape semantics. In Proceedings of the Audio Mostly 2015 on Interaction With Sound, Thessaloniki, Greece, 7–9 October 2015 ; p. 5.
2. Piczak, K.J. ESC: Dataset for environmental sound classification. In Proceedings of the 23rd ACM international conference on Multimedia, Brisbane, Australia, 26–30 October 2015 ; pp. 1015–1018.
3. Salamon, J.; Jacoby, C.; Bello, J.P. A dataset and taxonomy for urban sound research. In Proceedings of the 22nd ACM international conference on Multimedia, Orlando, FL, USA, 3–7 November 2014; pp. 1041–1044.
4. Zhan, Y.; Kuroda, T. Wearable sensor-based human activity recognition from environmental background sounds. *J. Ambient. Intell. Humaniz. Comput.* **2014**, *5*, 77–89.
5. Mendoza, J.M.; Tan, V.; Fuentes, V.; Perez, G.; Tiglao, N.M. Audio Event Detection Using Wireless Sensor Networks Based on Deep Learning. In *International Wireless Internet Conference*; Taipei, Taiwan 15–16 October 2018; Springer: Berlin, Germany; pp. 105–115.
6. Jakob, A.; Marco, G.; Stephanie, K.; Robert, G.; Christian, K.; Tobias, C.; Hanna, L. A Distributed Sensor Network for Monitoring Noise Level and Noise Sources in Urban Environments. In Proceedings of the 2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud), Barcelona, Spain, 6–8 August 2018; pp. 318–324.
7. Siamwala, A.; Lochhead, Z.; Abdulla, W. Environmental Noise Monitoring Using Distributed IoT Sensor Nodes. In Proceedings of the IEEE 2019 International Conference on Electronics, Information, and Communication (ICEIC), Beijing, China, 12–14 July 2019; pp. 1–10.
8. Kumari, S.; Roy, D.; Cartwright, M.; Bello, J.P.; Arora, A. EdgeLˆ 3: Compressing Lˆ 3-Net for Mote Scale Urban Noise Monitoring. In Proceedings of the 2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Rio de Janeiro, Brazil, 20–24 May 2019; pp. 877–884.
9. Bello, J.P.; Silva, C.; Nov, O.; DuBois, R.L.; Arora, A.; Salamon, J.; Mydlarz, C.; Doraiswamy, H. SONYC: A system for the monitoring, analysis and mitigation of urban noise pollution. *arXiv* **2018**, arXiv:1805.00889
10. Mydlarz, C.; Sharma, M.; Lockerman, Y.; Steers, B.; Silva, C.; Bello, J.P. The life of a New York City noise sensor network. *Sensors* **2019**, *19*, 1415.
11. Font, F.; Roma, G.; Serra, X. Freesound technical demo. In Proceedings of the 21st ACM international conference on Multimedia, Barcelona, Spain, 21–25 October 2013; pp. 411–412.
12. Blackman, R.B.; Tukey, J.W. The measurement of power spectra from the point of view of communications engineering—Part I. *Bell Syst. Tech. J.* **1958**, *37*, 185–282.
13. Hall, M.; Frank, E.; Holmes, G.; Pfahringer, B.; Reutemann, P.; Witten, I.H. The WEKA data mining software: an update. *Acm Sigkdd Explor. Newsl.* **2009**, *11*, 10–18.
14. Boddapati, V.; Petef, A.; Rasmusson, J.; Lundberg, L. Classifying environmental sounds using image recognition networks. *Procedia Comput. Sci.* **2017**, *112*, 2048–2056.
15. Cao, J.; Cao, M.; Wang, J.; Yin, C.; Wang, D.; Vidal, P.P. Urban noise recognition with convolutional neural network. *Multimed. Tools Appl.* **2018**, *78*, 29021–29041.
16. Su, Y.; Zhang, K.; Wang, J.; Madani, K. Environment Sound Classification Using a Two-Stream CNN Based on Decision-Level Fusion. *Sensors* **2019**, *19*, 1733.
17. Piczak, K.J. Environmental sound classification with convolutional neural networks. In Proceedings of the 2015 IEEE 25th International Workshop on Machine Learning for Signal Processing (MLSP), Boston, MA, USA, 17–20 September 2015; pp. 1–6.
18. Bogdanov, D.; Wack, N.; Gómez Gutiérrez, E.; Gulati, S.; Herrera Boyer, P.; Mayor, O.; Roma Trepat, G.; Salamon, J.; Zapata González, J.R.; Serra, X. Essentia: An audio analysis library for music information retrieval. In Proceedings of the 14th Conference of the International Society for Music Information Retrieval (ISMIR), Curitiba, Brazil, 4–8 November 2013; pp. 493–498.
19. pyAudioAnalysis. Available online: https://github.com/tyiannak/pyAudioAnalysis (accessed on 1 June 2019).

20. Giannakopoulos, T. pyaudioanalysis: An open-source Python library for audio signal analysis. *PLoS ONE* **2015**, *10*, e0144610.
21. Yaafe 0.64. Available online: http://yaafe.sourceforge.net/ (accessed on 1 June 2019).
22. Essentia 2.1. Available online: https://essentia.upf.edu/documentation/index.html (accessed on 1 June 2019).
23. aubio 0.4.0. Available online: http://aubio.org/ (accessed on 1 June 2019).
24. CLAM 1.4.0. Available online: http://clam-project.org/ (accessed on 1 June 2019).
25. LibROSA 0.6.3. Available online: https://librosa.github.io/librosa/ (accessed on 1 June 2019).
26. McFee, B.; Raffel, C.; Liang, D.; Ellis, D.P.; McVicar, M.; Battenberg, E.; Nieto, O. librosa: Audio and music signal analysis in Python. In Proceedings of the 14th Python in Science Conference, Austin, TX, USA, 6–12 July 2015; pp. 18–25.
27. Matlab Audio Analysis Library. Available online: https://www.mathworks.com/matlabcentral/fileexchange/45831-matlab-audio-analysis-library (accessed on 1 June 2019).
28. PyCASP. Available online: https://github.com/egonina/pycasp/ (accessed on 1 June 2019).
29. Gonina, E.I. A Framework for Productive, Efficient and Portable Parallel Computing. PhD thesis, EECS Department, University of California, Berkeley, 2013.
30. Seewave 2.1.3. Available online: https://cran.r-project.org/web/packages/seewave/index.html (accessed on 1 June 2019).
31. Bob 2.1.3. Available online: https://www.idiap.ch/software/bob/ (accessed on 1 June 2019).
32. Anjos, A.; Shafey, L.E.; Wallace, R.; Günther, M.; McCool, C.; Marcel, S. Bob: A free signal processing and machine learning toolbox for researchers. In Proceedings of the 20th ACM Conference on Multimedia Systems (ACMMM), Nara, Japan, 29 October–2 November 2012.
33. Scikit-learn 0.21.2. Available online: https://scikit-learn.org/stable/ (accessed on 1 June 2019).
34. Cowling, M.; Sitte, R. Comparison of techniques for environmental sound recognition. *Pattern Recognit. Lett.* **2003**, *24*, 2895–2907.
35. Detection and Classification of Acoustic Scenes and Events (DCASE), Challenge 2019, Task 5: Urban Sound Tagging. Available online: http://dcase.community/challenge2019/task-urban-sound-tagging (accessed on 7 August 2019).
36. Jekic, N.; Pester, A. Environmental Sound Recognition with Classical Machine Learning Algorithms. In *International Conference on Remote Engineering and Virtual Instrumentation*; Duesseldorf, Germany, 21–23 March 2018l; Springer: Berlin, Germany; pp. 14–21.
37. FreeSound: collaborative database of creative-commons licensed sound for musicians and sound lovers. Available online: https://freesound.org/ (accessed on 1 September 2019).
38. FreeSoundEffects Available online: https://www.freesoundeffects.com/free-sounds/airplane-10004/ (accessed on 1 September 2019).
39. Cass, S. Taking AI to the edge: Google's TPU now comes in a maker-friendly package. *IEEE Spectr.* **2019**, *56*, 16–17.